www.icgst.com

# In Situ Detection of Highway Lane Boundaries: GreedyHaarSpiker vs. SlopeInterceptFilter: A Comparison of Two Lane Boundary Detection Algorithms

Vladimir A. Kulyukin, Ashwani Chahal
*Department of Computer Science, Utah State University, 4205 Old Main Hill, Logan, UT 84322, USA*
[vladimir.kulyukin, ashwani.chahal]@aggiemail.usu.edu

## Abstract

An empirical comparison is presented of two algorithms for in situ vision-based detection of highway lane boundaries on a raspberry pi computer with a raspberry pi camera. The raspberry pi computer with a touchscreen is placed inside a Jeep Wrangler on the dashboard, and is powered through a 12V-to-5V car charger. The computer is connected to the camera which is mounted on a tripod next to the windshield. Two algorithms, *GreedyHaarSpiker* and *SlopeInterceptFilter*, are presented and empirically compared on videos captured by driving the car on a Northern Utah highway in different weather conditions. GreedyHaarSpiker is based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows. SlopeInterceptFilter applies probabilistic Hough Transform, filters detected lines by slope, and uses lane boundaries detected in previous frames to reduce the number of false positives. Both algorithms are implemented in Python 2.7.9 with OpenCV 3.0.0. To ensure the reproducibility of the results, the source code of the algorithms and the captured video data used in the experiments have been made public.

***Keywords:*** *Computer Vision, Lane Detection, Autonomous Vehicles, Self-Driving Cars, Wavelets, Probabilistic Hough Transform*

***Nomenclature:*** *CV - Computer Vision, AV - Autonomous Vehicle, HWT - Haar Wavelet Transform, HT - Hough Transform, LDWS - Lane Departure Warning System*

## 1 Introduction

Autonomous vehicles (AVs), also known as self-driving cars, are vehicles capable of making navigation decisions without human involvement. In September 2012, the Association of Electrical and Electronics Engineers (IEEE) has estimated that by 2040 up to 75% of all vehicles will be autonomous [1]. AVs have featured prominently in many research and commercial projects for decades. Several companies, such as Google, Audi, and BMW, have made considerable investments in the AV market, and are currently testing self-driving cars.

Proponents of self-driving cars argue that safety is a key public benefit of AV adoption, because more than 90 percent of road accidents are caused by human error [2]. Since all self-driving cars will use the same standardized algorithms, they will act predictably and in unison with respect to each other thereby reducing numbers of road accidents and recovery insurance claims [3]. Aged and disabled individuals will benefit from AVs due to enhanced human-independent mobility and decreased reliance on public transportation. AVs will likely reduce fuel consumption and commuting time and provide for better city planning, because many cities will not have to satisfy ever increasing demands for street and lot parking [4]. Some researchers argue that self-driving cars are expected to reduce congestion by using GPS and other localization technologies to build most efficient routes taking into account traffic jams and road closures [5].

While opponents of self-driving cars contend that self-driving cars may reduce numbers of traffic accidents, they point out that they will likely lead to loss of privacy and increased risks of hacking attacks and terrorism [6]. Another concern is that if self-driving cars are not adopted universally or, at least, very widely, accidents will still happen and many legal issues will be difficult to resolve. Self-driving cars may not operate well in all weather conditions. For example, in heavy rain lasers mounted on self-driving cars may seriously malfunction [7]. Broad adoption of self-driving cars will likely result in job losses among drivers and personal injury lawyers and cause many driver's education programs to go out of business. Some researchers argue that lack of stress during driving and more productive time on the road may create additional incentives to live even further from cities, which will increase the carbon footprint

of motor transportation systems [8]. Finally, many people who enjoy driving for the sake of driving, e.g., off-road enthusiasts, will unlikely trade their cars for AVs even when they are convinced of the fuel economy or safety benefits provided by AVs.

While we believe that self-driving cars may become a reality in the long term, provided that not only technical failures [9, 10] but also social and legal implications [11] of AV adoption are addressed, human drivers are, and will remain indispensable in the short and medium terms. Therefore, we believe it is important to seek solutions that enhance human drivers' safety. Vision-based lane boundary detection is one such enhancement. Vision-based lane detection modules may become an integral part of autopilots in cars and semi-trucks to improve the drivers' safety. Such autopilots will be similar to the ones in aircraft and ships and will keep the human in the loop in that the decision to engage or disengage the autopilot will be under the driver's control.

In this article, we compare two algorithms, *GreedyHaarSpiker* and *SlopeInterceptFilter*, developed in our laboratory for in situ vision-based detection of marked highway lane boundaries on a raspberry pi computer with a raspberry pi camera. GreedyHaarSpiker (GHS) [12] is an algorithm based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows. SlopeInterceptFilter (SIF) is an algorithm that applies probabilistic Hough Transform (HT), filters detected lines by slope, and uses previous frames to reduce the number of false positives. It is assumed that the lane boundaries are marked with white or yellow lines, as is the case on highways in many countries. The computer-camera unit is placed inside a 2016 Jeep Wrangler, next to the windshield, and is powered through a 12V-to-5V car charger. Both algorithms are implemented in Python 2.7.9 and OpenCV 3.0.0.

Our article is organized as follows. In Section 2, related work is reviewed. In Section 3, we describe the two algorithms, GHS and SIF, for in situ lane boundary detection. In Section 4, we present our experiments that compare the two algorithms on three highway drives. Our conclusions are presented in Section 5.

## 2 Related Work

The CMU Navigation Laboratory (Navlab) has built a series of robot cars, SUVs, and buses since 1984. The latest robotic car, Navlab 11, is a robot Jeep Wrangler equipped with a range of sensors for obstacle avoidance, path planning and following, and pedestrian detection [13]. The European Technology Platform on Smart Systems Integration project has reported significant contributions to collision avoidance, fleet management, autonomous cruise control, and cooperative driving [14]. In the U.S., both Google and Tesla have been commercializing their self-driving platforms [3, 15] over the past several years.

Shu and Tan [16] propose a novel method for tracking road lanes for vision-guided AV navigation. The researchers use inverse perspective mapping to remove the perspective from the camera and to detect the edges of the road lanes in inverse perspective mapped images. Particle filtering is used to compute to estimate real location of road lane lines in images.

Mandlik and Deshmukh [17] present a lane departure detection system (LDWS) for an Advanced Driver Assistance System (ADAS) to warn the driver of lane departures. The LDWS is based on a lane identification and tracking algorithm that uses the canny edge detector [18] and Hough transform (HT) [19] to detect vehicle lane departure on a raspberry pi computer. The longest straight lines are identified as the lane lines. The experiments with the system were conducted on images captured with a Intex IT-305WC webcam mounted on top of a toy vehicle. The captured images were sent wirelessly to an off-board Intel Core i3 1.80 GHz personal computer for processing.

Lim et al. [20] develop a linear parabolic lane model to localize lane boundaries in images. A CCD camera is used to capture video frames that are stored in the video port buffer of a TMS320DM642 DSP board. The proposed model is used to construct the geometry of a lane and to update the lane's parameters with Kalman filtering. Horizon localization is used to separate sky pixels and road pixels. To recognize lane markings, road pixels are subsequently removed from the road region.

Du et al. [21] propose a vision-based method to address such lane detection challenges as shadows, shifting lighting conditions, and faded-away lane lines. A ridge detector is used to pool lane line pixels. After a captured road image is de-noised, a sequential random sample consensus is employed to ensure that each lane line in the image is adequately sampled. In the final step, parallelism reinforcement is employed to enhance the lane model's accuracy. The researchers claim that their model is also fit to localize vehicles with respect to detected road lane lines.

Truong and Lee [22] present an algorithm to detect and estimate the curvature of lane boundaries. A vector lane concept and non-uniform B-spline (NUBS) interpolation method is used to construct the boundaries of road lane lines. Based on the lane boundary, the curvature of left and right lane boundaries are calculated. For experimental purposes, images are captured using a monocular camera.

Assidiq et al. [23] develop a vision-based lane detection approach to handle frequently varying lighting and shadow conditions. The framework acquires the frontal view from a camera mounted on the vehicle. Two hyperbolas, fitting to the edges of the lane, are paired with HT to extract the lane lines. It has also been asserted that the proposed lane detection frame-


work can be used on painted, unpainted, curved, and straight roads.

Wang et al. [24] propose a B-Snake based lane detection and tracking model for a range of lane structures. An algorithm, called *CHEVP*, is developed for providing initial positions for the B-Snake model. A minimum error method is proposed to determine the control points of the B-Snake model by the image forces on both sides of a lane. Experimental results suggest that the algorithm is robust against noise, shadows, and illumination variations in captured images of marked and unmarked roads.

Kim [25] presents a lane-detection-and-tracking algorithm to detect lane curvatures, lane changes, and splitting lanes. The detected lane markings are grouped into separate left and right lane-boundary hypotheses to handle merging and splitting lanes. The hypotheses are evaluated and grouped with a probabilistic, Markov-style framework.

Hsiao et al. [26] propose an embedded real-time LDWS for daytime and nighttime driving. The LDWS features a lane detection algorithm based on peak finding for feature extraction to detect lane boundaries. Gaussian smoothing and global edge detection are applied to reduce noise in images. The reported lane detection rates were 99.57% during the day and 98.88% at night on a sample of highway images.

Erickson and Landberg [27] propose a lane detection algorithm that combines HT [19] with a parabolic second degree fitting for curvature detection. On a raspberry pi 2, the algorithm's performance was found to be inadequate for high speed driving. However, when the object detection is removed from the algorithm, the algorithm meets the real time performance requirements on the raspberry pi.

Kulyukin [12] proposes an algorithm for in situ vision-based detection of highway lane boundaries on a raspberry pi computer coupled to a raspberry pi camera. The algorithm, called GreedyHaarSpiker (GHS), is based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows. We use this algorithm as one of the two algorithms for lane boundary detection in this investigation. The algorithm was also implemented in Python 2.7.9 and OpenCV 3.0.0 and was tested on the previous version of the video processing hardware unit (see Section 3 below) and installed in the same vehicle.

# 3 Two Algorithms

In this section, we describe the two algorithms, GHS and SIF, for in situ lane boundary detection. Both algorithms run on the same hardware installed inside a 2016 Jeep Wrangler, as shown in figures 1 and 2. The hardware consists of a wooden board, a touchscreen display, a raspberry pi computer, a pi camera, and a tripod. The pi computer is connected to the touchscreen and is attached with small screws to a metallic frame behind the touchscreen. The touchscreen with the pi computer behind it is attached to a wooden board with two metallic brackets, and is placed on the Jeep's dashboard near the middle of the windshield. Unlike in our previous two investigations [12, 28] on lane boundary detection, in this version of the video processing hardware, the pi camera facing the road is placed on a tripod perpendicular to the dashboard for stability. The hardware unit is powered with a 12V-to-5V car charger in the Jeep. When detected, lane boundaries are displayed in real time on the touchscreen. For this investigation, we used a raspberry pi 3.0 and a pi camera 1.3.

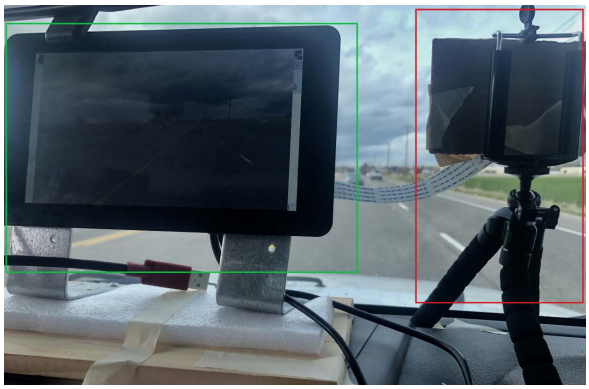

Figure 1: The driver's view of the lane boundary detection hardware: A touchscreen display (green box) with a raspberry pi computer behind it is placed on a wooden board mounted in the middle of the Jeep's dashboard; a pi camera (red box) is placed on a tripod to face the road

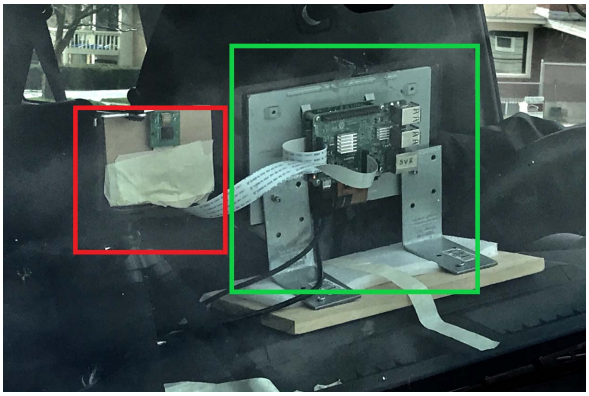

Figure 2: Outside view of the lane boundary detection hardware: A pi camera (red box) is placed on a tripod to face the road and is attached to the raspberry pi behind the touchscreen (green box)

## 3.1 GreedyHaarSpiker

The GHS algorithm is based on the concept of the 1D Haar Wavelet Spike that we proposed in our previous

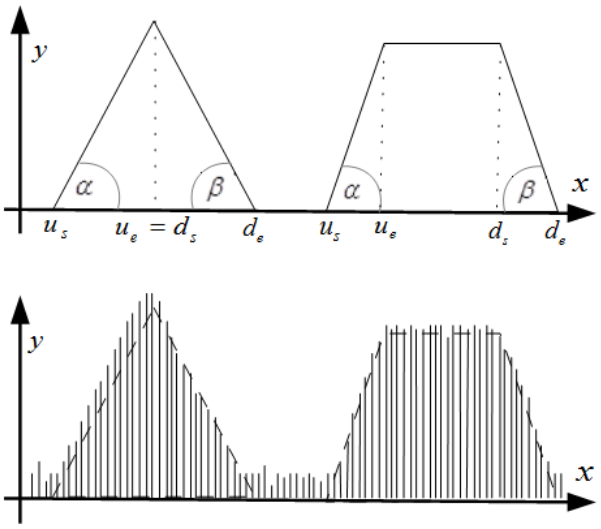


Figure 3: Two types of up-down spikes (above) and the corresponding Haar wavelets at a given scale $k$ from a signal (below).

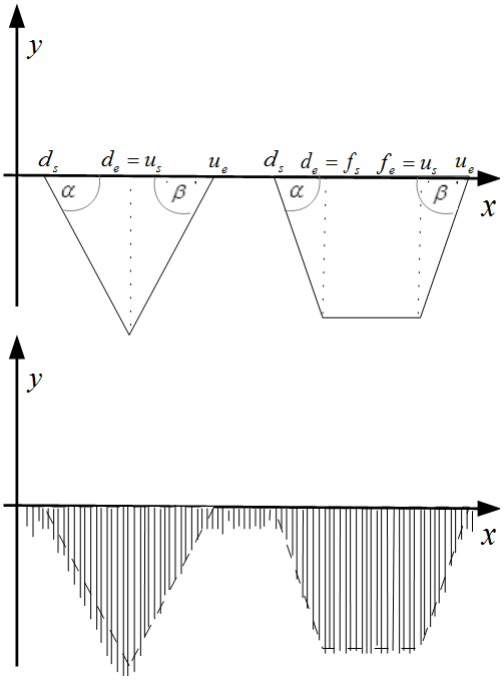


Figure 4: Two types of down-up spikes (above) and the corresponding Haar wavelets at a chosen scale $k$ from a signal (below).

research [12]. In this section, we summarize this concept and the algorithm. Interested readers can refer to our previous publications for more details (e.g., [28]).

In the 1D Haar Wavelet Transform (1D HWT), a signal is a vector in $R^n, n = 2^k, k \in N$. The signal's values may first rise and then fall or they may first fall and then rise. The signal's values may also have a relatively flat segment between the rise and the fall or the fall and the rise. There exist four types of spikes: up-down triangle, up-down trapezoid, down-up triangle, and down-up trapezoid. The difference between up-down and down-up spikes is the relative positions of the climb and decline segments. In trapezoid spikes, flat segments are always in between the climb and decline segments, regardless of their relative positions. Triangle spikes are trapezoid spikes with no flat segments.

Figure 3 shows up-down triangle and trapezoid spikes. Figure 4 shows down-up triangle and down-up trapezoid spikes. In both figures, the lower graphs represent the possible values of the corresponding Haar wavelets at a chosen scale $k$. Up-down spikes describe signals that first increase and then, after an optional flat segment, decrease. Down-up spikes describe signals that first decrease and then, after an optional flat segment, increase. Formally, a spike $\mathcal{S}$ is a nine element tuple whose elements are real numbers given in (1).

$$\mathcal{S} = (u_s, u_e, \alpha, f_s, f_e, \gamma, d_s, d_e, \beta) \tag{1}$$

The first two elements, $u_s$ and $u_e$, are the abscissae of the start and end of the spike's climb segment $[u_s, u_e]$, respectively, on which the wavelet coefficients of the 1D HWT increase. If $\omega_{u_s}^{(k)}$ and $\omega_{u_e}^{(k)}$ are the $k$-th scale wavelet coefficient ordinates at $u_s$ and $u_e$, respectively, the steepness of the climb, denoted by $\alpha$, is given in (2).

$$\alpha = tan^{-1}(u_e - u_s, \omega_{u_e}^{(k)} - \omega_{u_s}^{(k)}) \tag{2}$$

As one can see in figures 3 and 4, in (1), the flat segments of up-down or down-up spikes, are described by $f_s$, $f_e$, and $\gamma$, where $f_s$ and $f_e$ in (1) are the abscissae of the start and end of the spike's flat segment, respectively, over which the wavelet coefficients either remain at the same ordinate or have minor ordinate fluctuations.

If $\omega_{f_s}^{(k)}$ and $\omega_{f_e}^{(k)}$ are the $k$-th scale wavelet coefficients corresponding to $f_s$ and $f_e$, respectively, the spike's flatness, denoted by $\gamma$, is defined in (3).

$$\gamma = tan^{-1}(f_e - f_s, \omega_{f_e}^{(k)} - \omega_{f_s}^{(k)}) \tag{3}$$

The numbers of $d_s$ and $d_e$ in (1) are the absissae of the start and end, respectively, of the spike's decline segment $[d_s, d_e]$, over which the wavelet coefficients of the 1D HWT decrease.

If $\omega_{d_s}^{(k)}$ and $\omega_{d_e}^{(k)}$ are the $k$-th scale wavelet coefficient ordinates at $d_s$ and $d_e$, respectively, the steepness of the decline, denoted by $\beta$, is given in (4).

$$\beta = tan^{-1}(d_e - d_s, \omega_{d_e}^{(k)} - \omega_{d_s}^{(k)}) \tag{4}$$

The procedure *GHS* in Algorithm 1 presents the pseudocode of GreedyHaarSpiker. This procedure takes an RGB image, crops a region of interest (ROI) from it. The cropped image can be as large as the

Figure 5: Sample input image to GHS

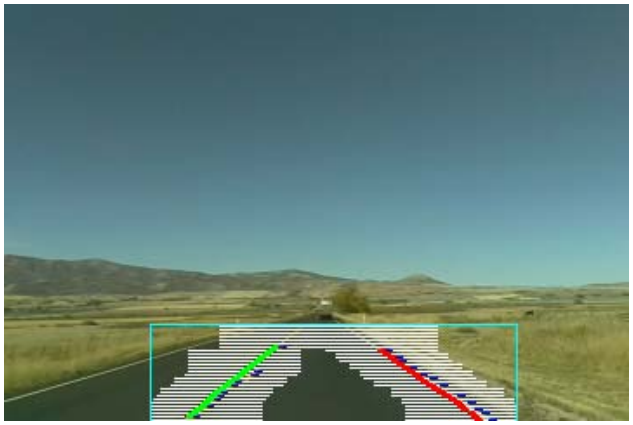

Figure 6: A region of interest (ROI) with scanlines and detected lane boundaries

original image or include a specific area in front of a vehicle, as shown in Figure 6. The cropped ROI is grayscaled, blurred with the $7 \times 7$ Gaussian kernel, and thresholded with the Otsu thresholding operator. The procedure *detectSpikes* is applied to this pre-processed ROI. This procedure returns two lists, LPoints and RPoints, of $(x, y)$ tuples. The procedure *fitLine* uses linear regression to fit a line through LPoints and RPoints. The lines are filtered by slope to reduce false positives. The slope thresholds for the left boundary are from -60° to -30°; the slope thresholds for the right boundary are from 30° to 60°. If the line through LPoints passes the left slope threshold filter, it is taken to be the left boundary of the vehicle's lane. If the line through RPoints passes the right slope treshold filter, it is taken to be the right boundary of the vehicle's lane.

**Algorithm 1** *GHS*(Img)

```
ROI ← cropROI(Img);
ROI ← convertToGrayscale(ROI);
ROI ← gaussianBlur(ROI);
ROI ← thresholdOTSU(ROI);
LPoints, RPoints ← detectSpikes(ROI);
LeftLaneBoundary ← fitLine(ROI, LPoints);
RightLaneBoundary ← fitLine(ROI, RPoints);
return LeftLaneBoundary, RightLaneBoundary;
```

**Algorithm 2** *detectHaarSpikes*(ROI, $s_r$, $e_r$, $\Delta$)

```
LLPoints ← [ ];
RPoints ← [ ];
LSpike ← NULL;
RSpike ← NULL;
r ← e_r;
while r ≤ s_r do
    LLine ← getLeftScanLine(ROI, r, LSpike);
    RLine ← getRightScanLine(ROI, r, RSpike);
    LHWT ← ordHWT(LLine);
    RHWT ← ordHWT(RLine);
    LSpike ← detectSpike(LHWT);
    RSpike ← detectSpike(RHWT);
    if LSpike ≠ NULL then
        LPoints.add(LSpike.getMidPointOfClimb());
    end if
    if RSpike ≠ NULL then
        RPoints.add(RSpike.getMidPointOfClimb());
    end if
    r ← r + Δ;
end while
```

In addition to a ROI, the procedure *detectHaarSpikes* in Algorithm 2 takes three integer parameters $s_r$, $e_r$, and $\Delta$. The parameters $s_r$ and $e_r$ specify the start and end rows, respectively, in the ROI where the spikes are detected. The parameter $\Delta$ specifies a step value for generating the exact row numbers where spikes are detected. For example, if the algorithm is to detect spikes in the row range $[50, 40]$ with $\Delta = -2$, the sequence of rows that will be considered is $(50, 48, 46, 44, 42, 40)$. The spike detection starts from the lower rows that are closest to the vehicle and moves up to the the rows that are further away from the vehicle.

The variables LPoints and RPoints contain the $(x, y)$ tuples returned to *detectLanes* after line fitting with linear regression. The variables LSpike and RSpike contain two spikes detected in the ordered HWTs of the left and right scanlines, respectively.

In the **while**-loop of *greedyHaarSpiker*, two scanlines, LLine and RLine, of 64 pixels each are chosen on the left and right sides of the ROI in row $r$. The scanline's length, i.e., 64, can be changed through a global variable but it has to be equal to an integral power of 2. A value of 64 was experimentally found to result in optimal performance.

If the value of LSpike is NULL, the left scanline starts at column 0. Similarly, if the value of RSpike is NULL, the right scanline starts at column $w - 1$, where $w$ is the width of the ROI, which, in the current implementation, is equal to 200. If the value of LSpike is not NULL, which means that a spike was detected


in the previous row, the left scanline, saved in the LLine variable, is centered on the middle of the two ordinates of the detected spike's climb segment, i.e., the ordinates of $u_s$ and $u_e$ in equation 1. The flat and down segments are currently not taken into account in the algorithm. The right scanline is detected and saved in RLine in the same way except that the spike saved in RSpike is used. In figure 6, the scanlines are shown as horizontal white lines on the left and right sides of each row. As row number $r$ approaches the upper boundary of the ROI, the gap between the left and right scan lines becomes smaller.

The procedure *detectSpike* in the **while**-loop of the procedure *detectHaarSpikes* uses thresholds for the angles of the climb, flat, and decline spike segments, i.e., $\alpha$, $\gamma$, $\beta$, and returns the leftmost spike that clears the thresholds. In the current implementation, $\alpha = \beta = 60°$ and $\gamma = 5°$. In other words, the spikes whose climb or decline angles are less than $60°$ are filtered out, and flat segments are detected so long as consecutive wave coefficients fluctuate within $\pm 5°$ of 0. This algorithm is greedy in that it always returns exactly one leftmost spike in each left scanline and exactly one leftmost spike in each right scanline. All other spikes are ignored. If no spikes clear the angle thresholds, the value of NULL is returned.

When **while**-loop of *greedyHaarSpiker* finishes, the lists LPoints and RPoints contain $(x, y)$ tuples representing the mid points of the climb segments of spikes detected in the left and right scanlines in each of the processed rows. These points are used by the procedure *fitLine* in *detectLanes* to fit lines through them. The lines identify the left and right lane boundaries.

## 3.2 SlopeInterceptFilter

**Algorithm 3** *SIF*(Img, $r$, $\theta$, $ll$, $lr$, $rl$, $rr$)

```
1: LHist ← [ ];
2: RHist ← [ ];
3: ROI ← cropROI(Img, r);
4: ROI ← convertToGrayscale(ROI);
5: ROI ← gaussianBlur(ROI);
6: ROI ← sobelEdgeDetector(ROI);
7: L = ← HoughTransform(ROI);
8: LL, RL ← slopeFilter(L, ll, lr, rl, rr);
9: LL, RL ← interceptFilter(LL, RL, θ);
10: if LLines == NULL then LL ← LHist;
11: if RLines == NULL then RL ← RHist;
12: LHist ← LL;
13: RHist ← RL;
14: return LL, RL;
```

The SIF algorithm also works on frames extracted from videos captured by the hardware unit described at the beginning of this section. The pseudocode for SIF is given in Algorithm 3. Each $1920 \times 1080$ frame is resized to be $400 \times 255$. A sample resized image is shown in figure 7. Since the road surface is always located in the lower part of each captured image, the image is divided horizontally with an input parameter $r$ specifying a row at which the division is performed ($r = 200$ in our case). The region above the specified row is discarded. The lower part is the ROI used in subsequent processing. The ROI is grayscaled and blurred with a Gaussian $5 \times 5$ kernel. The resulting image, after discarding the upper part and grayscaling and blurring the ROI, is shown in figure 8. The Sobel edge detector [29] is used on the ROI to detect edges, as shown in figure 9. Once the edges are detected in the image, HT [19] is applied to the ROI to detect lines. Figure 10 shows the lines detected in the image in figure 9.

Figure 7: A sample input image to SIF

Figure 8: Input image after discarding the upper part and grayscaling and blurring the lower part (i.e., the ROI)

As shown in figure 10, not all lines detected by HT belong to lane boundaries. To discard unwanted lines, the detected lines are filtered by slopes and intercepts. Since the actual lane boundaries, when present, are located in the bottom half of the image, a legal left lane boundary line makes a positive angle with the $x$-axis (i.e., it has a positive slope) and a legal right lane boundary line makes a negative angle with the $x$-axis (i.e., it has a negative slope). By analyzing a large sample of captured frames, we determined that the left lane border lines make an angle between $+30°$ to $+70°$ with the bottom edge of the image (i.e., the

Figure 9: ROI with detected edge pixels

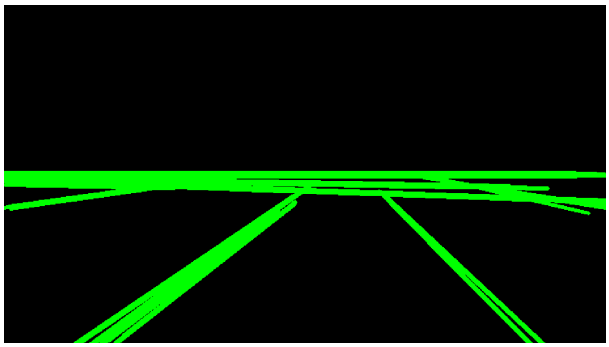

Figure 10: ROI with detected lines using Hough Transform prior to slope and intercept filtering

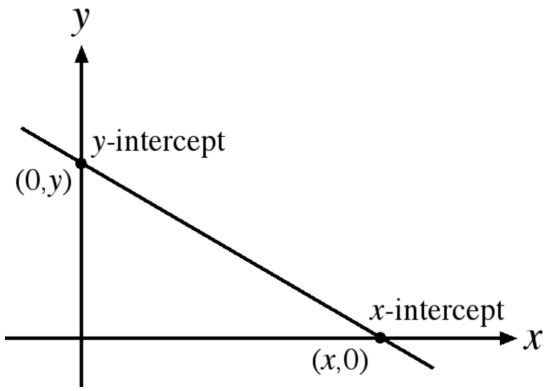


Figure 11: Finding y intercept of a line

$x$-axis) while the right lane boundary lines make an angle between $-30°$ to $-70°$ with the bottom edge. All lines whose slopes fall outside of these two ranges are discarded and the lines whose slopes fall within these ranges are labeled as potential left or right lane boundary lines, respectively.

A limitation of filtering lane lines on the basis of slope alone is that this filtering method considers all parallel lines (i.e., lines with the same slopes) as potential lane lines. Therefore, in addition to slope, another parameter, the $y$-intercept value of $c$ in $y = mx + c$, is taken into account to determine the exact lane boundary line position. Toward that end, the $y$-intercept value $c = y - m \cdot 0$ for each slope-filtered

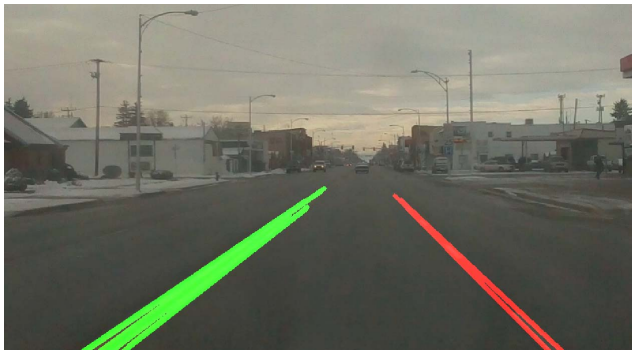

Figure 12: Lane boundary lines detected in the image shown in figure 7

line is calculated (see figure 11). The lines whose intercept values are not within $\theta$ pixels of the median $c$ value are discarded. In the current implementation, $\theta = 50$. Figure 12 shows the resulting lane boundary lines filtered by slope and intercept from the lines in figure 10 and drawn on the original input image in figure 7.

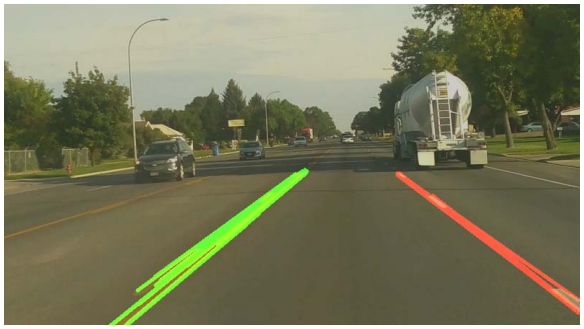

Figure 13: Both types of boundaries are detected and saved

Figure 14: The left boundary is detected but the right boundary (painted with yellow in the image) is restored from the right boundary lines detected in the image in figure 13

To reduce the number of false negatives (i.e., the images where the lane boundaries are not detected), the positions of the left and right lane boundaries detected in the current frame are saved. If the lane

boundaries are detected in the next frame, they are saved to replace the previously detected lane boundary lines. If no left lane boundary lines are detected in the current frame, the left boundary lines detected in the previous frame are assumed to be detected in the current frame. If no right lane boundary lines are detected in the current frame, the left lane boundary lines detected in the previous frame are assumed to be detected in the current frame.

The application of this technique of using a history of one frame is currently limited to ten consecutive frames. In other words, if the previously detected frames are used for ten consecutive frames (i.e., when the fresh lane boundary lines are not detected in ten consecutive frames), the algorithm fails. For example, if the vehicle leaves a highway for a country road, the absence of lanes will be detected in a matter of seconds. On highways, where painted lane boundary lines are more or less continuous, this technique works well to compensate for situations when lane lines are temporarily covered by passing vehicles, foreign objects, shadows, snow, etc. Figure 13 shows a frame where both left and right lane boundary lines are detected and successfully saved. Figure 14 shows a frame captured three frames after the frame in figure 13 where the right lane boundary lines were not detected due to a break in road paint. Therefore, the right lane boundary lines were restored from the right lane boundary lines detected in the frame in figure 13.

# 4 Experiments

The video data for the experiments were obtained with the raspberry pi hardware unit described in Section 3 on Highway 91 between Logan, Utah and Preston, Idaho. The pi camera's resolution was set to $1920 \times 1080$. The frame rate was set to 10. We drove the Jeep Wrangler on Highway 91 at a speed of 40 - 60 miles per hour on three separate days. To obtain data on a snowy day, we drove the Jeep on February 10, 2018 for 30 minutes. This drive took place after a snowfall but the highway was clear with most lanes visible to the human eye. To obtain data on a cloudy and rainy day, we drove the Jeep on April 7, 2018 for 30 minutes. This drive was on a cloudy day with light rain. To obtain data on a sunny day, we drove the Jeep on April 27, 2018 for approximately 10 minutes. The drive took place on a sunny day with clear skies and good visibility. Our drive on a sunny day was shorter, because our tripod malfunctioned and could not support the pi camera strictly perpendicular to the dashboard.

The videos were captured in h264 format and segmented into individual frames. The final data set which we used for evaluation included 33,892 images: 16,486 snowy images, 13,606 rainy images, and 3,800 sunny images. To ensure the reproducibility of our findings, we made public both our video data set [30] and our source code [31, 32].

Table 1: Performance of SIF in three different weather conditions

| Sets | Num. Images | TP | FP | Acc |
|---|---|---|---|---|
| Rainy | 3800 | 3711 | 89 | 97.65% |
| Sunny | 6139 | 5996 | 143 | 96.67% |
| Snowy | 16486 | 16255 | 231 | 98.59% |
| Total | 26425 | 25962 | 463 | 98.24% |

Table 1 gives the performance of the SIF algorithm on the captured data sets where the third, fourth, and fifth columns tabulate the percentages of true positives, false positives, and the overall accuracy. We evaluated the algorithm's performance by comparing the lane boundaries drawn in each image by the algorithm with the actual lane boundaries we saw in the same image. An image was evaluated as a true positive when both lanes were accurately detected (see figure 15). An image was evaluated as a false positive when both lane boundaries or one of the lane boundaries were inaccurately detected (see figure 16). An actual lane boundary was considered detected accurately if the boundary lines drawn by the algorithm were exactly aligned with the actual lane boundary. The SIF algorithm performed on par on all three data sets and recognized both lane boundaries in 98.24% of all the images.

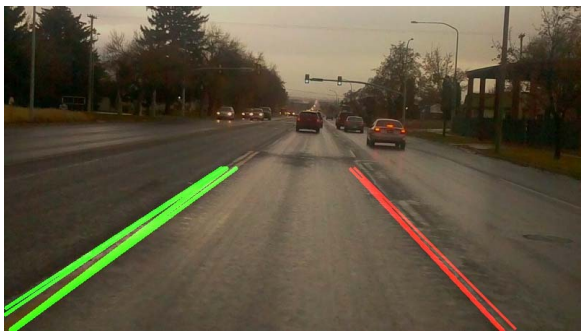

Figure 15: A true positive from the rainy data set where both boundaries are detected

We compared the performance of the SIF algorithm with the GHS algorithm [28]. We took three random samples of 120 images from 33,892 images and manually evaluated all images by first counting as true positives the images when both boundaries were detected and then counting as true positives the images when at least one lane boundary was detected. Tables 2 and 3 give the results of our evaluation.

In each random set, SIF outperformed GHS. The average SIF accuracies for detecting both boundaries was to 95.55%, and was 100% for detecting at least one boundary. The same average accuracies of GHS for the same categories were 50% and 84%, respectively. When we analyzed the results, we discovered

Figure 16: A false positive from the snowy data set: right boundary is detected accurately; left boundary is detected inaccurately

Table 2: GHS vs. SIF on three random sets: true positives are images with both boundaries detected

| Algo | Image Set | TP | Acc |
|---|---|---|---|
| GHS | set 1 | 58 | 48% |
| SIF | set 1 | 114 | 95% |
| GHS | set 2 | 59 | 49.16% |
| SIF | set 2 | 116 | 96.67% |
| GHS | set 3 | 63 | 52.5% |
| SIF | set 3 | 114 | 95% |

Table 3: GHS vs. SIF on three random sets: true positives are images with at least one boundary detected

| Algo | Image Set | TP | Acc |
|---|---|---|---|
| GHS | set 1 | 102 | 85% |
| SIF | set 1 | 120 | 100% |
| GHS | set 2 | 100 | 83.33% |
| SIF | set 2 | 120 | 100% |
| GHS | set 3 | 102 | 85% |
| SIF | set 3 | 120 | 100% |

that a principal reason for SIF outperforming GHS was its use of the previous frame to detect the boundary in the current frame even when it was not clearly present. Since GHS relies only on the current frame to detect boundaries, it sometimes missed one or two boundaries.

SIF also turns out to be a little faster than GHS in that it processes 7 frames per second on a raspberry pi 3.0 Model B ARMv8 1 GB RAM whereas GHS processes 4.5 frames per second. If a vehicle drives at a speed of $s$ mph, the distance between two consecutive frames is $d = s/(f \cdot 3600)$ miles, where $f$ is the number of frames per second. Thus, when a car travels at a speed of 60 mph, this distance is approximately 0.0024 miles for SIF and 0.0037 miles for GHS. Interested readers may refer to a video of a test drive on a Highway 89 from Logan, Utah to Bear Lake, Idaho [33]. This section of Highway 89 has many curves and shows that SIF handles curved lanes quite robustly.

# 5 Conclusions

In this article, we made an empirical comparison of two algorithms, SlopeInterceptFilter (SIF) and GreedyHaarSpiker (GHS), for in situ vision-based detection of highway lane boundaries. SIF applies probabilistic Hough Transform, filters detected lines by slope, and uses previous frames to reduce the number of false positives. GHS is a vision-based detection algorithm based on the detection of 1D Haar Wavelet spikes in 1D Ordered Haar Wavelet Transforms of image rows.

Both algorithms run on a raspberry pi 3.0 model B ARMv8 1 GB RAM connected to a pi camera 1.3 and a 7-inch touch screen display. This hardware is placed inside the car and the camera is placed next the windshield to capture the video. The raspberry pi and the display screen are powered via a standard 12V-to-5V (2 Amp) car charger. The hardware costs of the entire system is approximately 150 USD, which makes it possible for other researchers and practitioners to build our hardware unit and replicate our results.

The experimental video data for the empirical comparison were obtained with the above hardware unit on Highway 91 between Logan, Utah and Preston, Idaho. We drove a 2016 Jeep Wrangler at a speed of 40 - 60 miles per hour on three separate days to obtain videos in sunny, rainy, and snowy weather. Our experiments indicate that SIF, a much simpler algorithm, outperformed GHS in all weather conditions. A principal reason for SIF outperforming GHS is its use of the lane boundaries detected in a previous frame to detect the boundaries in the current frame. To ensure the reproducibility of the results, we have made publicly available both the source code of the algorithms used in our investigation and the captured video data.

# Acknowledgements

We are grateful to Prateek Vats for his help with road tests and data curation.

# References

[1] F. Tardo and M. Stickel. News release. IEEE, September 2012.

[2] National Highway Traffic Safety Administration. *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey.* U.S. Department of Transportation, February 2015.

[3] T. Simonite. Data shows google's robot cars are smoother, safer drivers than you or i. *MIT Technology Review*, October 2013.



[4] D. Ticoll. *Driving Changes: Automated Vehicles in Toronto*. University of Toronto Transportation Research Institute (UTTRI), October 2015.

[5] R.E. Stern, S. Cui, M. L. Delle Monache, R. Bhadani, M. Bunting, M. Churchill, N. Hamilton, R. Haulcy, H. Pohlmanng, F. Wu, B. Piccoli, B. Seibold, J. Sprinkled, and D.B. Work. Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments. *Transportation Research Part C: Emerging Technologies*, 89:205–221, 2018.

[6] Q. Miller. Robotic cars and their new crime paradigms. *LinkedIn Pulse*, September 3, 2013.

[7] N. E. Boudette. 5 things that give self-driving cars headaches. *The New York Times*, June 4, 2016.

[8] M. Ufberg. Whoops: The self-driving tesla may make us love urban sprawl again. *Wired*, October 10, 2015.

[9] D. Yadron and D. Tynan. Tesla driver dies in first fatal crash while using autopilot mode. *Guardian*, July 1, 2016.

[10] V. Mathur. Google autonomous car experiences another crash. *Government Technology*, July 17, 2015.

[11] J. Boeglin. The costs of self-driving cars: reconciling freedom and privacy with tort liability in autonomous vehicle regulation. *Yale Journal of Law and Technology*, 17(1):Article 4, 2015.

[12] V. Kulyukin. Greedyhaarspiker: an algorithm for in situ detection of highway lane boundaries with 1d haar wavelet spikes. *Graphics, Vision and Image Processing (GVIP)*, 17(2):9 – 17, 11 2017.

[13] S. Thrun. Toward robotic cars. *Communications of the ACM*, 53(4):99 – 106, 2010.

[14] J. Dokic, B. Müller, and G. Meyer. *European Roadmap Smart Systems for Automated Driving*. European Technology Platform on Smart System Integration, Berlin, Germany, 2015.

[15] G. Nelson. Tesla beams down 'autopilot' mode to model s. *Automative News*, October 14, 2015.

[16] Y. Shu and Z. Tan. Vision based lane detection in autonomous vehicle. In *Proceedings of the Fifth World Congress on Intelligent Control and Automation*, pages 5258 – 5260, Hangzhou, China, June 15 – 19 2004.

[17] P. Mandlik and A. Deshmukh. Raspberry pi based real time lane departure warning system using image processing. *International Journal of Engineering Research and Technology*, 5(6):755 – 762, 2016.

[18] J.F. Canny. A computational approach to edge detection. *IEEE Trans. on Pat. Anal. And Mach. Intel.*, 8:679 – 688, 1986.

[19] R.O. Duda and P.E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Comm. ACM*, 15:11 – 15, 1972.

[20] K. H. Lim, K. P. Seng, A. C. L. Ngo, and L. M. Ang. Real-time implementation of vision-based lane detection and tracking. In *International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, pages 364 – 367, Hangzhou, Zhejiang, China, 26 – 27 August 2009.

[21] X. Du and K. K. Tan. Vision-based approach towards lane line detection and vehicle localization. *Machine Vision and Applications*, 27:175 – 191, 2015.

[22] Q. B. Truong and B. R. Lee. New lane detection algorithm for autonomous vehicles using computer vision. In *International Conference on Control, Automation and Systems*, pages 1208 – 1213, Seoul, South Korea, 14 – 17 October 2008.

[23] A. Assidiq, O. O. Khalifa, R. Islam, and S. Khan. Real time lane detection for autonomous vehicles. In *International Conference on Computer and Communication Engineering*, pages 82 – 88, Kuala Lumpur, Malaysia, 13 – 15 May 2008.

[24] Y. Wang, E. Teoha, and D. Shen. Lane detection and tracking using b-snake. *Image and Vision Computing*, 22:269 – 280, 2008.

[25] Z. Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Trans. on Intelligent Transportation Systems*, 9(1):16 – 26, 2008.

[26] P. Hsiao, C. Yeh, S. Huang, and L. Fu. A portable vision-based real-time lane departure warning system: day and night. *IEEE Trans. on Vehicular Technology*, 58(4):2089 – 2094, 2009.

[27] J. Eriksson and J. Landberg. Lane departure warning and object detection through sensor fusion of cellphone data. Technical report, Department of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden, 2015.

[28] V. Kulyukin and V.R. Sudini. Real-time vision-based lane detection on raspberry pi with 1d haar wavelet spikes. In *Lecture Notes in Engineering and Computer Science: Proceedings of the International MultiConference of Engineers and Computer Scientists*, pages 75 – 80, Hong Kong, China, March 2017.



[29] I. Sobel. An isotropic 3 x 3 image gradient operator. *Machine vision for three-dimensional scene*, pages 75 – 80, 1990.

[30] V. Kulyukin, A. Chahal, and P. Vats. Video data captured on highway 91 in northern utah. `https://www.dropbox.com/sh/s6nuctxzlxb1jew/AADJZh31kjjeyjDCVPFND9ska?dl=0`, February 2017.

[31] V. Kulyukin and V. R. Sudini. Source code of ghs algorithm. `https://github.com/VKEDCO/PYPL/tree/master/haar_spiker`, February 2017.

[32] A. Chahal and V. Kulyukin. Source code of sif algorithm. `https://github.com/ashwani27chahal/LaneDetectionResearch.git`, April 2018.

[33] V. Kulyukin, A. Chahal, and P. Vats. Testing sif algorithm on curved sections of highway 89. `https://www.youtube.com/watch?v=1EdCja8NtUo`, February 2017.

# Biographies

**Vladimir A. Kulyukin** is an Associate Professor of Computer Science at Utah State University. He holds a Ph.D. in Computer Science from the University of Chicago that he received in 1998. His research interests include AI, computer vision, and sensor fusion.

**Ashwani Chahal** received an M.S. in Computer Science at Utah State University in 2018 where he worked under the supervision of Dr. Kulyukin. His research interests include computer vision and software engineering.